The PCL Theorem. 
Transactions cannot be Parallel, Consistent and Live.

Victor Bushkov
EPFL, IC, LPD
victor.bushkov@epfl.ch

Panagiota Fatourou *
University of Crete &
FORTH-ICS
faturu@csd.uoc.gr

Dmytro Dziuma
FORTH-ICS
ddziuma@ics.forth.gr

Rachid Guerraoui
EPFL, IC, LPD
rachid.guerraoui@epfl.ch

ABSTRACT

We show that it is impossible to design a transactional memory system which ensures parallelism, i.e., transactions do not need to synchronize unless they access the same application objects, while ensuring very little consistency, i.e., a consistency condition, called weak adaptive consistency, introduced here and which is weaker than snapshot isolation, processor consistency, and any other consistency condition stronger than them (such as opacity, serializability, causal serializability, etc.), and very little liveness, i.e., that transactions eventually commit if they run solo.

Categories and Subject Descriptors

D.1.3 [Programming Techniques]: Concurrent Programming

Keywords

transactional memory; disjoint-access-parallelism; snapshot isolation; processor consistency; weak adaptive consistency; obstruction-freedom; lower bounds; universal constructions

1. INTRODUCTION

The paradigm of transactions [20, 26, 35] is appealing for its simplicity but implementing it efficiently is challenging. Ideally a transactional system should not introduce any contention between transactions beyond that inherently due to the actual code of the transactions. In other words, if two transactions access disjoint sets of data items, then none of these transactions should delay the other one, i.e., these transactions should not contend on any base object. This requirement has been called strict disjoint-access-parallelism [2].

*Currently with École Polytechnique Fédérale de Lausanne (EPFL), Switzerland, as an EcoCloud visiting professor.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

SPAA '14 Prague, Czech Republic
Copyright 2014 ACM 978-1-4503-2821-0/14/06 ...$15.00.
http://dx.doi.org/10.1145/2612669.2612690.

[22]. Base objects are low-level objects, which typically provide atomic primitives like read/write, load linked/store conditional, compare-and-swap, used to implement transactional systems. Two transactions contend on some base object if both access that object during their executions and one of them performs a non-trivial operation on that object, i.e., an operation which updates its state.

Strict disjoint access parallelism can be ensured by blocking transactional memory (TM) systems; indeed, TL [14], a lock-based TM algorithm, ensures strict disjoint-access-parallelism and strict serializability [39]. It was shown in [21] that a strictly disjoint-access-parallel TM algorithm cannot ensure both obstruction-freedom (i.e. a weak non-blocking liveness condition) and serializability (i.e. a consistency condition weaker than strict serializability). Specifically, obstruction-freedom [25] ensures that a transaction is aborted only if step contention is encountered during the course of its execution. Serializability [30] ensures that, in any execution, all committed transactions (and some that have not completed yet) execute like in a legal sequential execution.

In this paper, we study the following question: can we ensure strict disjoint-access-parallelism and obstruction freedom if we weaken safety? In other words, is serializability indeed a major factor against strong parallelism? We focus on a new weak consistency condition that we introduce in this paper, called weak adaptive consistency. Weak adaptive consistency is weaker than (a weak variant of) snapshot isolation [10] and processor consistency [19]. Thus, it is weaker than serializability, causal serializability and all other consistency conditions that are stronger than processor consistency (or snapshot isolation or even the union of both). Our PCL theorem states that even with weak safety and weak liveness, the described task is still impossible: specifically, it is not possible to implement a transactional memory system which ensures strict disjoint-access-parallelism (Parallelism), weak adaptive consistency (Consistency), and obstruction-freedom (Liveness).

Weak adaptive consistency weakens snapshot isolation in two ways: (1) each process is allowed to have its own sequential view and (2) it is possible to partition the transactions of an execution in such a way that each set of transactions in the partition satisfies either snapshot isolation or processor consistency. Snapshot isolation [10] requires that transactions should be executed as if every read operation observes a consistent snapshot of the memory that was taken when
the transaction started. To make our result stronger, in our definition of snapshot isolation, we do not require the extra constraint (met in the literature [10, 16, 33] for snapshot isolation) that from two concurrent transactions writing to the same data item, only one can commit, and we do not impose any restriction on the value that a read on some data item \( x \) by a transaction \( T \) may return if \( T \) has written \( x \) before invoking this read. Processor consistency [19, 3] allows each process to have its own sequential view which should respect the process-order of writes, additionally it requires writes to the same data item appear in the same order in all sequential views. Processor consistency is stronger than PRAM consistency [28, 3], which does not require writes to the same data item to appear in the same order in all sequential views. Processor consistency is stronger than [8], which requires each sequential view to respect a relation on transactions, called causality relation.

The proof of our impossibility result is based on indistinguishability arguments. The main difficulty comes from the fact that the read operations of a transaction do not have to be serialized at the same point as its write operations. Basically, snapshot isolation and especially weak adaptive consistency allow more executions to be correct and it is much harder to construct an execution which violates it. We end up constructing two legal executions where a transaction must read the same values for data items. We then prove that in one of these two executions this is not the case.

This paper is structured as follows. Section 2 gives an overview of the related work. Section 3 gives a system model and all necessary definitions. Section 4 gives the PCL theorem and its proof. Section 5 presents concluding remarks.

2. RELATED WORK

The notion of disjoint-access-parallelism appears in the literature [4, 8, 10, 23, 27, 31] in many flavors. Disjoint-access-parallelism was first introduced in [27] through the notion of conflicting transactions. Later variants [4, 8, 10] employed the concept of a conflict graph. A conflict graph is a graph whose nodes represent transactions (or operations) performed in an execution interval \( \alpha \) (i.e. the execution interval of those transactions overlap with \( \alpha \)) and an edge exists between two nodes if the corresponding transactions (operations) access the same data item in \( \alpha \) (i.e. they conflict in \( \alpha \)). In most of these definitions, disjoint-access-parallelism requires any two transactions to contend on a base object only if there is a path in the conflict graph of the minimal execution interval that contains the execution intervals of both transactions such that every two consecutive transactions in the path conflict. In [2, 8, 10, 27], additional constraints are placed on the length of the path in the conflict graph, resulting in what is known as \( d\)-local contention property, where \( d \) is an upper bound on the length of the path. In [27], where disjoint-access-parallelism originally appeared, an additional constraint on the step complexity of each operation was provided in the definition. Stronger versions of disjoint-access-parallelism usually result in more parallelism and therefore they are highly desirable when designing TM implementations. Weaker versions of disjoint-access-parallelism may result in less parallelism but are easier to implement.

Attiya et al. [8] proved that no disjoint-access-parallel TM implementation can support wait-free and invisible read-only transactions. A read-only transaction does not perform writes on data items; an invisible transaction does not perform non-trivial operations on base objects when reading data items. The variant of disjoint-access-parallelism considered in [8] stipulates that processes executing two transactions concurrently contend on a base object only if there is a path between the two transactions in the conflict graph. Although our impossibility does not hold for this variant of disjoint-access-parallelism, our impossibility result considers a much weaker liveness property and holds even for TM algorithms where read-only transactions are visible.

Recent work [7] proved that, if the TM algorithm does not have access to the code of each transaction, a property similar to wait-freedom, called local progress, cannot be ensured by any TM algorithm. In [10], it was proved that wait-freedom cannot be achieved even if this restriction is abandoned (given that each time a transaction aborts, it restarts its execution), if the TM algorithm ensures strict serializability and a weak version of disjoint-access-parallelism, called feeble disjoint-access-parallelism. Thus, one must consider weaker consistency or progress properties as we do here.

Pelerman et al. [31] proved that no disjoint-access-parallel TM algorithm can be strictly serializable and MV-permissive. The impossibility result holds under the assumptions that the TM algorithm does not have access to the code of transactions and the code for reading and writing data items terminates within a finite number of steps. Pelerman et al. [31] considered the same variant of disjoint-access-parallelism as in [8]. A TM implementation satisfies MV-permissiveness if a transaction aborts only if it is a write transaction that conflicts with another write transaction. This impossibility result can be beaten [7] if the stated assumptions do not hold. Our impossibility result holds if the TM ensures just weak snapshot isolation, even if it is MV-permissive.

Several software TM implementations [35, 11, 17, 29, 30, 25] are disjoint-access-parallel: TL [14] ensures strict disjoint-access-parallelism but is blocking since it uses locks; the rest satisfy weaker forms of disjoint-access-parallelism [8]. Among them OSTM [17] is lock-free. The TM in [35] is also lock-free but it has been designed for static transactions that access a pre-determined set of memory locations. Apparently, our impossibility result does not contradict these implementations because all of them, except TL, ensure weaker variants of disjoint-access-parallelism and some of them weaker progress as well. Also, our impossibility result does not contradict TL since TL uses locks and consequently does not ensure obstruction-freedom. Linearizable universal constructions [23, 24], which ensure some form of disjoint-access-parallelism, are presented in [1, 9, 15, 37]. Barnes [9] implementation is lock-free. The universal construction in [15] ensures wait-freedom when applied to objects that have a bound on the number of data items accessed by each operation they support, and lock-freedom in other cases. Disjoint-access-parallel wait-free universal constructions when each operation accesses a fixed number of predetermined memory locations are provided in [2, 37].

Snapshot isolation was originally introduced as an isolation level for database transactions [10, 16] to increase concurrency. In TM computing, snapshot isolation has been studied in [1, 15, 33, 34]. An STM algorithm, called SI-STM, which ensures snapshot isolation is presented in [33]. SI-STM employs a global clock mechanism and therefore, it is not disjoint-access-parallel. In [15], static analysis techniques are presented to detect, at compile time, consistency anomalies that may arise when
the TM algorithm satisfies snapshot isolation or other safety properties. Snapshot isolation on TM for message-passing systems has been studied in \cite{20}.

Our definition of snapshot isolation is weaker than that defined for database transactions \cite{14} for the following reasons. First, we do not put any constraint on the value returned by any read that occurs after a write to the same data item in the same transaction. Second, we do not place the “first committer wins” rule, i.e. we abandon the requirement to abort one out of two concurrent transactions that are writing to the same data item. By introducing these constraints, we would make our impossibility result weaker.

3. PRELIMINARIES

System. We consider an asynchronous system with $n$ processes which communicate by accessing shared base objects. A base object provides atomic primitives to access or modify its state. The system may support various types of base objects like read/write registers, CAS, etc. A primitive that does not change the state of an object is called trivial (otherwise it is called non-trivial).

Transactions. Transactional memory (TM) employs transactions to execute pieces of sequential code in a concurrent environment. Each piece of code contains accesses to pieces of data, called data items, that may be accessed by several processes when the code is executed concurrently; so TM should synchronize these accesses. To achieve this, a TM algorithm usually provides a shared representation for each data item by using base objects. A transaction may either commit, in which case all its updates become visible to other transactions, or abort and then its updates are discarded.

A TM algorithm provides implementations for the routines $x$.read(), which returns a value for $x$ if the operation was successful or $A_r$ if the transaction has to abort, and $x$.write($v$), which writes value $v$ to data item $x$ and returns ok if the write was successful or $A_w$ if the transaction has to abort. In addition, a TM algorithm provides implementations for the routines begin, which is called when a transaction starts and returns ok, commit, which is called when $T$ tries to commit and returns either $C_r$ (commit) or $A_r$ (abort), and abort, which aborts $T$ and returns $A_r$. Each time a transaction calls one of these routines we say that it invokes an operation; when the execution of the routine completes, a response is returned.

Executions and configurations. A configuration is a vector with components comprising the state of each process and the state of each base object. In an initial configuration, processes and base objects are in initial states. A step of a process consists of a single primitive on a single base object, the response to that primitive, and zero or more local operations that are performed after the access and which may cause the internal state of the process to change; each step is executed atomically. Invocations and responses performed by transactions are considered as steps. An execution $\alpha$ is a sequence of steps. An execution is legal starting from a configuration $C$ if the sequence of steps performed by each process follows the algorithm for that process (starting from its state in $C$) and, for each base object, the responses to the operations performed on the object are in accordance with its specification (and the state of the object at configuration $C$). We use $\alpha \cdot \beta$ to denote the execution $\alpha$ immediately followed by the execution $\beta$ and say that $\alpha$ is a prefix of $\alpha \cdot \beta$. An execution is solo if every step is performed by the same process. Two executions $\alpha_1$ and $\alpha_2$ starting from configurations $C_1$ and $C_2$, respectively, are indistinguishable to some process $p$, if the state of $p$ is the same in $C_1$ and $C_2$, and the sequence of steps performed by $p$ (and thus also the responses $p$ receives) are the same during both executions.

Transaction $T$ completes in $\alpha$, if $\alpha$ contains $C_T$ or $A_T$. Transaction $T$ accesses $x$ in $\alpha$, if $\alpha$ contains either $x$.write() or $x$.read(). The execution interval of a completed transaction $T$ in $\alpha$ is the subsequence of consecutive steps of $\alpha$ starting with the first step executed by any of the operations invoked by $T$ and ending with the last such step. The execution interval of a transaction $T$ that does not complete in $\alpha$ is the suffix of $\alpha$ starting with the first step executed by any of the operations invoked by $T$. The active execution interval of any transaction (completed or not) $T$ in $\alpha$ is the subsequence of consecutive steps of $\alpha$ starting with the first step executed by any of the operations invoked by $T$ and ending with the last such step. A TM algorithm is obstruction-free if a transaction $T$ can be aborted only when other processes take steps during the execution interval of $T$.

Histories. A history $H$ is a sequence of invocations and responses performed by transactions. Given an execution $\alpha$, we denote by $H_\alpha$ the sequence of invocations and responses performed by the transactions in $\alpha$. We denote by $H[T]$ the longest subsequence of $H$ consisting only of invocations and responses of a transaction $T$. Transaction $T$ is in history $H$ if $H[T]$ is not empty. History $H$ is well-formed if for every transaction $T$ in $H$ the following holds for $H[T]$:

(i) $H[T]$ is a sequence of alternating invocations and responses starting with $\text{begin}$-ok. (ii) Each read invocation is followed either by a value or by $A_T$. (iii) Each write invocation is followed by either an ok response or $A_T$. (iv) Each invocation of commit is followed by $C_T$ or $A_T$. (v) Each invocation of abort is followed by $A_T$. (vi) No invocation follows after $C_T$ or $A_T$. Herein, we consider only well-formed histories. We say that $T$ commits (aborts) in $H$ if $H[T]$ ends with $C_T$ ($A_T$). If $T$ does not commit or abort in $H$, then $T$ is live in $H$. If $H[T]$ ends with an invocation of commit, then $T$ is commit-pending. Transaction $T_1$ precedes transaction $T_2$ in execution $\alpha$ (denoted $T_1 \prec \alpha \cdot T_2$), if $T_1$ is not live in $H_\alpha$ and $A_{T_1} \prec \alpha \cdot C_{T_2}$ or $C_{T_1} \prec \alpha \cdot A_{T_2}$. If $T_1 \prec \alpha \cdot T_2$ and $T_2 \prec \alpha \cdot T_1$, then $T_1$ and $T_2$ are concurrent in $\alpha$.

A history $H$ is sequential if no two transactions are concurrent in $H$. $H$ is complete if it does not contain any live transactions. Transaction $T$ is legal in a sequential history $H$, if for every $x$.read() by $T$ which returns some value $v$ the following holds: (i) if $T$ executes an $x$.write() before $x$.read(), then $v$ is the argument of the last such $x$.write() invocation in $T$; otherwise, (ii) if there is an invocation of $x$.write() by some committed transaction that precedes $T$, then $v$ is the argument of the last such $x$.write() in $H$; otherwise (iii) $v$ is the initial value of $x$. A complete sequential history $H$ is legal if each transaction is legal in $H$.

Disjoint-access-parallelism. For proving the impossibility result presented in Section \cite{3}, we consider a collection of simple static transactions. Hence, to simplify the definitions in this paragraph, we assume that transactions are static and predefined\footnote{Apparentely, this assumption makes the impossibility result proved in Section \cite{3} stronger.}. i.e. we assume that the data items on which $T$ invokes read and write (in any execution con-}
taining $T$) are the same and can be derived by inspecting $T$’s code; we call the set of these data items the data set $D(T)$ of $T$. Note that the set of data items accessed by $T$ in a specific execution might be a proper subset of $D(T)$ if $T$ is not committed in this execution. For example, consider a transaction $T$ whose code implies that $T$ accesses data items $x$ and $y$ and let $\alpha$ be an execution in which transaction $T$ invokes $x.read()$ and gets $AR$ as a response; then, $D(T) = \{x, y\}$ but $T$ accesses only data item $x$ in $\alpha$.

We say that two transactions $T_1$ and $T_2$ conflict, if $D(T_1) \cap D(T_2) \neq \emptyset$. We say that two executions $\sigma$ contend on a base object $o$ if they both contain a primitive operation on $o$ and one of these primitive operations is non-trivial. Denote by $\alpha I$ the subsequence of $\alpha$ consisting of all steps executed by $T$. A TM implementation $I$ is strict disjoint-access-parallel, if in each execution $\alpha$ of $I$, and for every two transactions $T_1$ and $T_2$ executed in $\alpha$, $\alpha[T_1]$ and $\alpha[T_2]$ contend on some base object, only if $T_1$ and $T_2$ conflict.

**Consistency.** A read operation $x.read()$ by some transaction $T$ is global if $T$ has not invoked $x.write()$ before invoking $x.read()$. Let $T$ be a committed or commit-pending transaction executed by a process $p_i$ in a history $H$. Let $T|read$ be the longest subsequence of $H[T]$ consisting only of global read invocations and their corresponding responses and $T|write$ be the longest subsequence of $H[T]$ consisting only of write invocations and their corresponding responses. Let $\lambda$ denote the empty history. Then we define transactions $T_{gr}$ and $T_w$ (both executed by $p_i$) in the following way:

1. $T_{gr} = T|read \cdot commit_{T_{gr}} \cdot C_{T_{gr}}$ if $T|read \neq \lambda$, and $T_{gr} = \lambda$ otherwise, and
2. $T_w = T|write \cdot commit_{T_w} \cdot C_{T_w}$ if $T|write \neq \lambda$, and $T_w = \lambda$ otherwise.

**Definition 3.1 (Snapshot isolation).** An execution $\alpha$ satisfies snapshot isolation, if (i) there exists a set $com(\alpha)$ consisting of all committed and some of the commit-pending transactions in $\alpha$ and (ii) it is possible to insert (in $\alpha$) a serialization point $\tau_T$ for each transaction $T \in com(\alpha)$, so that if $\sigma_\alpha$ is the sequence defined by these serialization points, the following holds:

1. $\tau_T \prec \sigma_\alpha$,
2. both $\tau_T$ and $\sigma_\alpha$ are inserted within the active execution interval of $T$,
3. if $H_{\sigma_\alpha}$ is the history we get by replacing each $\tau_T$ in $\sigma_\alpha$, with $H[T]$, if $T$ commits in $\alpha$, or with $H[T \cdot C_T\tau_T]$, if $T$ is commit-pending in $\alpha$, then every transaction executed by $p_i$ is legal in $H_{\sigma_\alpha}$.

Let $T_1$ and $T_2$ be two transactions in an execution $\alpha$ such that either $T_1$ and $T_2$ are the same or the invocation of begin$_{T_1}$ precedes the invocation of begin$_{T_2}$. A consistency group $G(T_1, T_2)$ of $\alpha$ is a set of transactions from $\alpha$ such that:

1. $T_1$, $T_2$ belong to $G(T_1, T_2)$, and
2. a transaction $T_k$ belongs to $G(T_1, T_2)$ if the invocation of begin$_{T_k}$ occurs in $\alpha$ between the invocation of begin$_{T_1}$ and the invocation of begin$_{T_2}$.

In other words, a consistency group $G(T_1, T_2)$ of $\alpha$ is a set containing the transactions that start their execution between the beginning of $T_1$ and the beginning of $T_2$ (inclusive). An active execution interval of $G(T_1, T_2)$ is the longest execution interval of $\alpha$ which includes all steps in $\alpha$ from the first step of $T_1$ to the last step of any transaction from $G(T_1, T_2)$. A consistency group $G(T_1, T_2)$ precedes a consistency group $G(T'_1, T'_2)$ in $\alpha$, if the last step of any transaction from $G(T_1, T_2)$ precedes the first step of $T'_1$ in $\alpha$.

A consistency partition $P(\alpha)$ of an execution $\alpha$ is a sequence $G(T_{1,1}, T_{1,1}), G(T_{1,2}, T_{1,2}), \ldots, G(T_{1,n}, T_{1,n})$ of consistency groups such that:

1. $T_{1,1}$ is the transaction which invokes the first begin in $\alpha$ and $T_{1,n}$ is the transaction which invokes the last begin in $\alpha$,
2. $\forall k \in \{1, \ldots, n\}$, either $T_{1,k} = T_{r,k}$ or begin$_{T_{1,k}} \prec$ begin$_{T_{r,k}}$ in $\alpha$,
3. $\forall k \in \{1, \ldots, n-1\}$, begin$_{T_{1,k}} \prec$ begin$_{T_{1,k+1}}$ in $\alpha$, and there is no transaction that invokes begin between begin$_{T_{1,k}}$ and begin$_{T_{1,k+1}}$ in $\alpha$.

**Definition 3.3 (Weak adaptive consistency).** An execution $\alpha$ satisfies weak adaptive consistency if it is possible to do all of the following: (i) choose a consistency partition $P(\alpha)$, (ii) partition all groups in $P(\alpha)$ into two disjoint sets of groups: a set $SI(P(\alpha))$ of snapshot isolation groups and a set $PC(P(\alpha))$ of processor consistency groups, (iii) choose a set $com(\alpha)$ consisting of all committed and some of an execution; specifically, the transactions of the execution can be partitioned into groups so that either snapshot isolation or processor consistency is ensured for the transactions of each group. Processor consistency is a safety property which allows each process to have its own sequential view but requires that writes to the same data item occur in the same order in each sequential view.

**Definition 3.2 (Processor consistency).** An execution $\alpha$ is processor consistent if (i) there exists a set $com(\alpha)$ consisting of all committed and some of the commit-pending transactions in $\alpha$ and (ii) for each process $p_i$, it is possible to insert (in $\alpha$) a serialization point $\tau_T$ for each transaction $T \in com(\alpha)$, so that if $\sigma_{\alpha}^{p_i}$ is the sequence defined by these serialization points, the following holds:

1. $\forall T_1, T_2 \in com(\alpha)$:
   
   (a) if $T_1$ and $T_2$ are executed by the same process and $T_1 \prec T_2$, then $\tau_{T_1}$ precedes $\tau_{T_2}$ in $\sigma_{\alpha}^{p_i}$,
   
   (b) if $T_1$ and $T_2$ write to the same data item and $\tau_{T_1}$ precedes $\tau_{T_2}$ in $\sigma_{\alpha}^{p_i}$, then $\forall j \in \{1, \ldots, n\}$, $\tau_{T_1}$ precedes $\tau_{T_2}$ in $\sigma_{\alpha}^{p_i}$,

2. if $H_{\sigma_{\alpha}^{p_i}}$ is the history we get by replacing each $\tau_T$ in $\sigma_{\alpha}^{p_i}$ with $H[T]$, if $T$ commits in $\alpha$, or with $H[T \cdot C_T\tau_T]$, if $T$ is commit-pending in $\alpha$, then every transaction executed by $p_i$ is legal in $H_{\sigma_{\alpha}^{p_i}}$.
the commit-pending transactions in $\alpha$, and (iv) for each process $p$, insert (in $\alpha$) a global read serialization point $\tau_{r,gr}$ and a write serialization point $\tau_{r,w}$, for each transaction $T \in \text{com}(\alpha)$, so that if $\sigma^*_p$ is the sequence defined by these serialization points, the following holds:

1. $\tau_{r,gr}$ precedes $\tau_{r,w}$ in $\sigma^*_p$.
2. $\forall T_1, T_2 \in \text{com}(\alpha)$, if $T_1$ and $T_2$ write to the same data item and $\tau_{r,gr}$ precedes $\tau_{r,w}$ in $\sigma^*_p$, then $\forall j \in \{1, \ldots, n\}$, $\tau_{r,gr}$ precedes $\tau_{r,w}$ in $\sigma^*_p$.
3. for each group $G(T_i,k,T_{r,k}) \in S(P(\alpha))$ the following holds: $\forall T_m \in G(T_i,k,T_{r,k}) \cap \text{com}(\alpha)$, both $\tau_{m,gr}$ and $\tau_{m,w}$ are inserted within the active execution interval of $T_m$.
4. for each group $G(T_i,k,T_{r,k}) \in PC(P(\alpha))$ the following holds: $\forall T_m \in G(T_i,k,T_{r,k}) \cap \text{com}(\alpha)$, no other serialization point is inserted between $\tau_{m,gr}$ and $\tau_{m,w}$ and both $\tau_{m,gr}$ and $\tau_{m,w}$ are inserted within the active execution interval of $G(T_i,k,T_{r,k})$.
5. if $H_{\sigma^*_p}$ is the history we get by replacing each $\tau_{r,gr}$ with $T_{r,gr}$ and each $\tau_{r,w}$ with $T_{r,w}$ in $\sigma^*_p$, then every transaction executed by $p$ in $H_{\sigma^*_p}$ is legal in $H_{\sigma^*_p}$.

Consider an execution $\alpha$ that satisfies weak adaptive consistency, let $\sigma^*_w$ be the sequence of serialization points for process $p$. For simplicity, we use the following notation: $\tau_{r,gr}^{\alpha} <_i \tau_{r,w}^{\alpha}$, where $T, T' \in \text{com}(\alpha)$ and $l_1, l_2 \in (gr,w)$, to identify that $\tau_{r,gr}^{\alpha}$ precedes $\tau_{r,w}^{\alpha}$ in $\sigma^*_w$. We remark that items 3 and 4 of Definition 4.3 imply that the global read and write serialization points of any transaction $T \in \text{com}(\alpha)$ are placed within the active execution interval of the consistency group to which $T$ belongs.

An STM implementation $I$ satisfies weak adaptive consistency, if each of the executions produced by $I$ satisfies weak adaptive consistency. Weak adaptive consistency is weaker than processor consistency, and consequently is weaker than causal serializability, serializability, opacity and any other property stronger than processor consistency. This is so because if an execution $\alpha$ satisfies processor consistency, then there exists a consistency partition $P(\alpha) = G(T_i,T_r)$ consisting only of one processor consistency group such that the active execution interval of $G(T_i,T_r)$ is exactly $\alpha$, and therefore, serialization points of transactions from $G(T_i,T_r)$ can be inserted anywhere in $\alpha$. Weak adaptive consistency is weaker than snapshot isolation because in the definition of snapshot isolation there is only one sequential view $\sigma^*_w$, and condition 2 of the above definition trivially holds for the case of a single sequential view $\sigma^*_w$. In fact, weak adaptive consistency is even weaker than the union of snapshot isolation and processor consistency.

4. THE PCL THEOREM

In this section we show that it is impossible to implement a TM which ensures weak adaptive consistency, obstruction-freedom, and strict disjoint-access-parallelism. The main idea behind the proof is the following. We design two legal executions $\alpha = \alpha_1 \cdot \alpha_2 \cdot s_1 \cdot s_2 \cdot \alpha_3$ and $\alpha' = \alpha_1 \cdot \alpha_2 \cdot s_2 \cdot s_1 \cdot \alpha'_2$, where $\alpha_1$ and $\alpha_2$ are parts of solo executions of some transactions $T_1$ (executed by process $p_1$) and $T_2$ (executed by process $p_2$), respectively, $s_1$ and $s_2$ are single steps by $T_1$ and $T_2$, respectively, and $\alpha_2'$ and $\alpha'_2$ are solo executions of $T_7$ (executed by process $p_7$) until it commits. We prove that $s_1$ and $s_2$ are steps accessing different base objects, so $\alpha_2'$ is indistinguishable from $\alpha'_2$ to process $p_7$. We also prove that there exists a data item in $T_7$'s read set for which $T_7$ reads a different value in $\alpha_1$ from the value read for the same data item in $\alpha'_1$, which is a contradiction.

**Theorem 4.1 (The PCL Theorem).** There is no TM implementation which is strict disjoint-access-parallel and satisfies weak adaptive consistency and obstruction-freedom.

Proof. Assume, by contradiction, that there exists an obstruction-free implementation $I$ which is strict disjoint-access-parallel and satisfies weak adaptive consistency.

We use the following notation: we denote by $b_k, c_k, d_k$ data items written by transaction $T_k$ and by $e_{k,m}$ data items written by both transactions $T_k$ and $T_m$. Consider the following transactions (the initial value of every data item is considered to be 0):

- $T_1$, executed by process $p_1$, which reads data items $b_3$ and $b_7$, and writes the value 1 to data items $a_1, b_1, c_1, d_1, e_1, e_3, e_4$.
- $T_2$, executed by process $p_2$, which reads data items $b_5$ and $b_7$, and writes the value 2 to data items $a_1, b_2, c_2, d_2, e_2, e_5, e_7$.
- $T_3$, executed by process $p_3$, which reads data items $b_1$ and $b_4$, and writes the value 1 to data items $b_3, c_3, e_1, e_3, e_4$.
- $T_4$, executed by process $p_4$, which reads data items $d_2$ and $c_3$, and writes the value 1 to data items $b_4, e_3, e_4$.
- $T_5$, executed by process $p_5$, which reads data items $b_2$ and $b_6$, and writes the value 1 to data items $b_5, c_5, e_2, e_5, e_6, e_7$.
- $T_6$, executed by process $p_6$, which reads data items $d_1$ and $c_5$, and writes the value 1 to data items $b_6, e_5, e_6$.
- $T_7$, executed by process $p_7$, which reads data items $a_1, c_1, c_2$, and writes the value 1 to data items $b_7, e_2, e_7$.

**Definition of $\alpha_1$ and $s_1$.** Let transaction $T_1$ be executed solo from the initial configuration $C_0$. Because $p_1$ runs solo and $I$ is obstruction-free, $T_1$ eventually commits. In the resulting execution, $T_1$ reads the value 0 for data items $b_3$ and $b_7$ because $I$ satisfies weak adaptive consistency and there is no transaction that writes to these data items in this execution. Let $C'$ be the configuration resulting from the execution of the last step of $T_1$.

If $T_3$ is executed solo from the initial configuration $C_0$, then in the resulting execution, $T_3$ reads 0 for $b_1$ (since $I$ satisfies weak adaptive consistency and no transaction writes to $b_1$ in this execution).

Consider now the execution $\delta_1$ where transaction $T_3$ is executed solo from $C'$ until it commits. We prove that $T_3$ reads the value 1 for data item $b_1$ in $\delta_1$. Since $I$ satisfies weak adaptive consistency, there exists a consistency partition $P(\delta_1)$ which satisfies the conditions of Definition 4.3.

We consider the following cases:

- Assume first that $P(\delta_1) = G(T_1, T_3)$ and $SI(P(\delta_1)) = \{G(T_1, T_3)\}$. Since $G(T_1, T_3)$ is a snapshot isolation group, then $\tau_{r,gr}$ must be placed within the active execution interval of $T_1$ and $\tau_{r,gr}$ must be placed within
Figure 1: Executions $\alpha_1$, $\alpha_3$, $\alpha_5'$, and step $s_1$.

Figure 2: Executions $\alpha_1$, $\alpha_2$, $\alpha_5$, $\alpha_5'$, and step $s_2$.

Figure 3: Execution $\beta$.

Figure 4: Execution $\beta'$.

Figure 5: Values read by transactions in execution $\beta$. Where $x : v$ denotes a read from $x$ which returns value $v$, $x(v)$ denotes a write to $x$ which writes value $v$, and $\bullet$ denotes a commit event.

Figure 6: Values read by transactions in execution $\beta'$. 
the active execution interval of $T_{2}$. Since $T_{1} < s_{1}, T_{3}$, then $s_{1} < s_{3} + t_{5}$. It follows that $T_{3}$ must observe the update performed on data item $b_{1}$ by $T_{1}$, and consequently $T_{3}$ must read 1 for $b_{1}$.

1. Assume that $P(\delta_{1}) = G(T_{1}, T_{3})$ and $PCP(\delta_{1}) = \{G(T_{1}, T_{3})\}$. Because $T_{1}$ reads 0 for $b_{1}$ in $\delta_{1}$, it follows that $s_{1} < s_{3} + t_{5}$. Since $G(T_{1}, T_{3})$ is a processor consistency group, no serialization point is inserted between $s_{3} + t_{5}$ and $s_{3} + t_{5}$. Thus, $s_{1} < s_{3} + t_{5}$. Because $T_{1}$ and $T_{3}$ write to the same data item $c_{1}, a_{1}$, it follows that $s_{1} < s_{3} + t_{5}$. Since no serialization point is inserted between $s_{3} + t_{5}$ and $s_{3} + t_{5}$, it follows that $s_{1} < s_{3} + t_{5}$, so $T_{3}$ must read 1 for $b_{1}$.

2. Assume now that $P(\delta_{1}) = G(T_{1}, T_{1}), G(T_{2}, T_{3})$. Because $T_{1} < s_{1} < T_{3}$, it follows that $G(T_{1}, T_{1})$ precedes $G(T_{2}, T_{3})$ in $\delta_{1}$. Since $T_{1}$ and $T_{3}$ should be placed within the execution interval of $G(T_{1}, T_{1})$ and $s_{1}$ is the sole execution of $T_{1}$ from configuration $C_{1}$ until it commits, there is a step $s_{2}$, resulting in a configuration $C_{2}$, such that:

1. if $a_{2}$ is the sole execution of $T_{2}$ from configuration $C_{2}$, where $C_{2}$ is the configuration just before the execution of $s_{2}$, then $T_{2}$ reads 0 for $b_{2}$ in $a_{2}$;
2. if $a_{3}$ is the sole execution of $T_{3}$ from configuration $C_{2}$, then $T_{3}$ reads 2 for $b_{2}$ and 0 for $b_{3}$ in $a_{3}$;
3. if $a_{4}$ is the sole execution of $T_{1}$ from configuration $C_{2}$ until $C_{2}$ is reached (Figure 2), then $T_{2}$ invokes $\text{commit}_{T_{2}}$ in $a_{2}$.

4. $T_{2}$ reads the value 0 for data items $b_{5}$ and $b_{7}$ in $a_{2}$.
5. $s_{2}$ applies a non-trivial operation on some base object $a_{2}$ which is read in $a_{3}$ and $a_{2}$.

Claim 3: $a_{1} \neq a_{2}$

Proof: Assume that $a_{1} = a_{2}$. Consider an execution $a_{1} \cdot a_{2} \cdot s_{1} \cdot \gamma_{3}$, where $s_{1}$ is a single step by $p_{1}$ and $\gamma_{3}$ is a solo execution of $T_{3}$ by $p_{3}$ until $T_{3}$ commits. We argue that $s_{1} = s_{1} \vee s_{1} = \gamma_{3}$ is indistinguishable to execution $a_{3}$ to $p_{3}$. Obviously, since $s_{1}$ is the step that $p_{1}$ is poised to perform after $a_{1}$, $s_{1}$ and $s_{1} = \gamma_{3}$ access the same base object, namely object $a_{1}$. Thus, if $s_{1}$ and $s_{1} = \gamma_{3}$ are different, they differ in their response.

Since $T_{3}$ and $T_{2}$ do not conflict, strict disjoint-access-parallelism implies that $a_{2}$ does not contain any non-trivial operation on base objects read in $\gamma_{3}$. Thus, the prefix of $a_{3}$ until the point that $a_{1}$ is first accessed is also a prefix of $\gamma_{3}$. Therefore, $T_{3}$ reads $a_{1}$ in $\gamma_{3}$ (as it does in $a_{3}$).

Because $\gamma_{3}$ reads $a_{1}$ and $T_{1}$ and $T_{2}$ do not conflict, strict disjoint-access-parallelism implies that $a_{1} = a_{3}$ does not contain a non-trivial operation on $a_{1} = a_{2}$. It follows that $s_{1} = s_{1} = \gamma_{3}$. This and the fact that $T_{1}$ and $T_{2}$ do not conflict imply that $\gamma_{3}$ is indistinguishable from $a_{1}$ to $p_{3}$. So, execution $a_{2} \cdot a_{2} \cdot s_{1} \cdot a_{3}$ is legal.

Since $p_{2}$ is poised to execute a step which applies a non-trivial operation on $a_{2} = a_{1}$ after $a_{1} \cdot a_{2} \cdot s_{1} \cdot a_{3}$ and $a_{1}$ is read in $a_{3}$, it follows that in execution $a_{2} \cdot a_{2} \cdot s_{1} \cdot a_{3} \cdot s_{2}$, where $s_{2}^{*}$ is a single step by $p_{2}$, strict disjoint-access-parallelism is violated. This is a contradiction. Thus, $a_{1} \neq a_{2}$.

Consider executions $a_{1} \cdot a_{2} \cdot s_{1} \cdot a_{3} \cdot a_{4}$ and $a_{1} \cdot a_{2} \cdot s_{1} \cdot a_{3} \cdot a_{4}$. Since $a_{4}$ is the only execution of $T_{4}$ until it commits. We first argue that $s_{2} = s_{2}^{*}$.

Recall that $a_{1} \cdot a_{2} \cdot s_{1} \cdot a_{3} \cdot a_{4}$ is legal. So $a_{1} \cdot a_{2} \cdot s_{1} \cdot a_{3} \cdot a_{4}$ is legal. Notice that $s_{2}$, like $a_{2}$, accesses base object $a_{2}$. It remains to argue that the response of $s_{2}^{*}$ is the same as that of $s_{2}$. Recall that $a_{3}$ does not contain a non-trivial operation on $a_{2}$. It remains to argue that the same is true for $a_{3}$.

Consider the execution $a_{2} = a_{1} \cdot a_{2} \cdot s_{1} \cdot a_{3} \cdot a_{4} \cdot s_{2}$. Recall that $a_{2}$ is the sole execution of $T_{2}$ from configuration $C_{2}$. Since $T_{2}$ does not conflict with $T_{1}, T_{3}$, and $T_{4}, T_{2}$, strict disjoint-access-parallelism implies that $a_{2} \cdot a_{2} \cdot s_{1} \cdot a_{3} \cdot a_{4} \cdot s_{2}$ is legal. Since $a_{4}$ reads $a_{2}$ and $T_{2}$ does not conflict with $T_{3}$, strict disjoint-access-parallelism implies that $a_{4}$ does not contain a non-trivial operation on $a_{2}$. It follows that $s_{2}^{*} = s_{2}^{*}$.

Let $\beta = a_{1} \cdot a_{2} \cdot s_{1} \cdot a_{3} \cdot a_{4} \cdot s_{2} \cdot a_{7}$ (see Figure 3). We first argue that $\beta$ is legal. This is so because $T_{2}$ does not conflict neither with $T_{3}$ nor with $T_{4}$, so $a_{7}$ does not access any base object modified in $a_{3}$ or $a_{4}$.

We now argue that $T_{2}$ reads 2 for data item $a_{2}$ and 1 for data item $c_{1}$ in $a_{7}$.

Claim 4: $T_{2}$ reads the value 2 for data items $a_{2}$ and $c_{2}$, and the value 1 for data item $c_{1}$ in $a_{7}$.

Proof: We first prove that transaction $T_{4}$ reads 0 for $d_{2}$ in $a_{4}$. Recall that $d_{2} = a_{1} \cdot a_{2} \cdot s_{1} \cdot a_{3} \cdot a_{4} \cdot a_{5}$ is legal. Since $I$ satisfies weak adaptive consistency, there exists a consistency partition $P(\delta_{2})$ and a set $\text{comm}(\delta_{2})$ of committed and commit-pending transactions from $\delta_{2}$ which satisfy the conditions of Definition 3.2. We argue that $T_{2} \notin \text{comm}(\delta_{2})$. For simplicity, throughout the proof, we use the term execution interval instead of active execution interval, whenever it is clear from the context.
Assume, by contradiction, that $T_2 \notin \text{com}(\delta_2)$. We consider the following cases.

- Assume first that $P(\delta_2)$ includes $G(T_i, T_3)$ for some $i \in \{3, 4, 5\}$. Since the execution intervals of $T_2$ and $T_1$ precede the execution intervals of $T_3$, $T_4$, and $T_5$, and the execution intervals of $T_3$, $T_4$, and $T_5$ do not overlap in $\delta_2$, it follows that the execution interval of the consistency group containing $T_2$ in $P(\delta_2)$ precedes the execution interval of $(T_i, T_3)$. Thus, $\tau_{T_2,w} <_5 \tau_{T_3,gr}$. This contradicts the fact that $T_2$ reads 0 for $b_2$ in $\alpha_5$.

- Assume that $P(\delta_2)$ includes $G(T_i, T_3)$ for some $i \in \{1, 2\}$ and $G(T_i, T_3) \in \text{SI}(P(\delta_2))$. Since $G(T_i, T_3)$ is a snapshot isolation group and the execution interval of $T_2$ precedes the execution interval of $T_3$ in $\delta_3$, it follows that $\tau_{T_2,w} <_5 \tau_{T_3,gr}$. This contradicts the fact that $T_2$ reads 0 for $b_2$ in $\alpha_5$.

- Assume that $P(\delta_2)$ includes $G(T_i, T_3)$ for some $i \in \{1, 2\}$ and $G(T_i, T_3) \in \text{PC}(P(\delta_2))$. Because $T_2$ reads 0 for $b_2$ in $\alpha_2$, it follows that $\tau_{T_2,w} <_5 \tau_{T_3,gr}$. Since no point is inserted between $\tau_{T_2,gr}$ and $\tau_{T_3,gr}$, it follows that $\tau_{T_2,w} <_5 \tau_{T_3,gr}$. Therefore, $T_2$ and $T_3$ write to the same data item $c_2$, it follows that $\tau_{T_2,w} <_5 \tau_{T_3,gr}$. Since no point is inserted between $\tau_{T_2,gr}$ and $\tau_{T_3,gr}$, it follows that $\tau_{T_2,w} <_5 \tau_{T_3,gr}$. This contradicts the fact that $T_2$ reads 0 for $b_2$ in $\alpha_5$.

Hence, $T_2 \notin \text{com}(\delta_2)$, and consequently $T_3$ reads 0 for $d_2$ in $\alpha_4$.

We next argue that $T_4$ reads 1 for $c_3$ in $\alpha_4$. Since $\delta_4 = \alpha_1 \cdot \alpha_2 \cdot s_1 \cdot \alpha_3 \cdot \alpha_4$ is legal and $I$ satisfies weak adaptive consistency, there exists a consistency partition $P(\delta_4)$ and a set $\text{com}(\delta_4)$ of committed and commit-pending transactions from $\delta_4$ which satisfy the conditions of Definition 33. We consider the following cases:

- Assume first that $P(\delta_4)$ includes $G(T_i, T_4)$ for some $i \in \{1, 2, 3\}$ and $G(T_i, T_4) \in \text{SI}(P(\delta_3))$. Since $T_3 \in \delta_3$, $T_3$ reads 0 for $b_3$ in $\alpha_3$, it follows that $\tau_{T_3,w} <_4 \tau_{T_4,gr}$. Since no point is inserted between $\tau_{T_3,gr}$ and $\tau_{T_4,gr}$, it follows that $\tau_{T_3,w} <_4 \tau_{T_4,gr}$. Because $T_3$ and $T_4$ write to the same data item $c_3$, it follows that $\tau_{T_3,w} <_4 \tau_{T_4,gr}$. Since no point is inserted between $\tau_{T_3,gr}$ and $\tau_{T_4,gr}$, it follows that $\tau_{T_3,w} <_4 \tau_{T_4,gr}$. We conclude that (in all cases) $\tau_{T_3,w} <_4 \tau_{T_4,gr}$. Thus, $T_4$ reads 1 for $c_3$ in $\alpha_4$.

We now prove that $T_7$ reads 2 for $c_2$ in $\alpha_7$. Notice that $\alpha_1 \cdot \alpha_2 \cdot s_1 \cdot s_2 \cdot \alpha_5$ is legal. This is so because $s_1$ is an object and $T_3$ does not conflict with $T_1$, so it does not access $a_1$ in $\alpha_5$. We argue that $\delta_5 = \alpha_1 \cdot \alpha_2 \cdot s_1 \cdot s_2 \cdot \alpha_5 \cdot \alpha_7$ is also legal. This is so because $\alpha_1 \cdot \alpha_2 \cdot s_1 \cdot s_2 \cdot \alpha_5$ and $\alpha_1 \cdot \alpha_2 \cdot s_1 \cdot s_2 \cdot \alpha_7$ are legal and $T_7$ does not conflict with $T_5$.

Since $I$ satisfies weak adaptive consistency, there exists a consistency partition $P(\delta_4)$ and a set $\text{com}(\delta_4)$ of committed and commit-pending transactions from $\delta_4$ which satisfy the conditions of Definition 33. We argue that $T_7 \notin \text{com}(\delta_5)$. We consider the following cases:

- Assume first that $P(\delta_4)$ includes $G(T_i, T_7)$ for some $i \in \{5, 7\}$. Since the execution intervals of $T_1$ and $T_2$ do not overlap with the execution intervals of $T_3$ and $T_7$ in $\delta_4$, it follows that the execution interval of the consistency group partition $P(\delta_4)$ that contains transaction $T_2$ precedes the execution interval of $G(T_i, T_7)$. Therefore, $\tau_{T_2,w} <_7 \tau_{T_7,gr}$, and consequently $T_7$ must read 2 for $c_7$ in $\alpha_7$.

- Assume now that $P(\delta_4)$ includes $G(T_i, T_7)$, for some $i \in \{1, 2\}$ and $G(T_i, T_7) \in \text{SI}(P(\delta_3))$. Since the execution interval of $T_2$ precedes the execution interval of $T_2$ and $T_7$, $T_7 \in G(T_i, T_7)$, which is a snapshot isolation group, it follows that $\tau_{T_2,w} <_7 \tau_{T_7,gr}$. Thus, $T_7$ must read 2 for $c_7$ in $\alpha_7$.

- Assume now that $P(\delta_4)$ includes $G(T_i, T_7)$, for some $i \in \{1, 2\}$ and $G(T_i, T_7) \in \text{PC}(P(\delta_3))$. Because $T_2$ reads 0 for $b_2$ in $\alpha_2$, it follows that $\tau_{T_2,w} <_7 \tau_{T_7,gr}$. Since no point is inserted between $\tau_{T_2,gr}$ and $\tau_{T_7,gr}$, it follows that $\tau_{T_2,w} <_7 \tau_{T_7,gr}$. Therefore, $T_2$ and $T_7$ write to the same data item $c_2$, it follows that $\tau_{T_2,w} <_7 \tau_{T_7,gr}$. Since no point is inserted between $\tau_{T_2,gr}$ and $\tau_{T_7,gr}$, it follows that $\tau_{T_2,w} <_7 \tau_{T_7,gr}$. Thus, $T_7$ must read 2 for $c_7$ in $\alpha_7$.
Assume, by contradiction, that $T_1 \in com(\delta_5)$. We consider the following cases.

- **Assume first that $P(\delta_5)$ includes $G(T_1, T_3)$ for some $i \in \{2, 3, 5, 6\}$. Since the execution interval of $T_1$ precedes the execution intervals of $T_2$, $T_3$, $T_6$ and $T_5$, and the executions intervals of $T_2$, $T_3$, $T_6$ and $T_5$ do not overlap in $\delta_5$, it follows that the execution interval of the consistency group containing $T_1$ precedes the execution interval of $G(T_1, T_3)$ in $\delta_5$. Thus, $\tau^1,w \prec \tau, gr$. This contradicts the fact that $T_3$ reads 0 for $b_1$ in $\alpha_3$.**

- **Assume now that $P(\delta_5) = G(T_1, T_3)$ and that it holds that $SI(P(\delta_5)) = \{G(T_1, T_3)\}$. Since the execution interval of $T_1$ precedes the execution interval of $T_3$ in $\delta_5$, it follows that $\tau, w \prec_3 \tau, gr$. This contradicts the fact that $T_3$ reads 0 for $b_1$ in $\alpha_3$.**

- **Assume now that $P(\delta_5) = G(T_1, T_3)$ and that it holds that $PC(P(\delta_5)) = \{G(T_1, T_3)\}$. Because $T_1$ reads 0 for $b_3$ in $\alpha_1$, it follows that $\tau_{1, gr} \prec_1 \tau_{3, w}$. Since no point is inserted between $\tau_{1, gr}$ and $\tau_{3, w}$, it follows that $\tau_{3, w} \prec \tau_{3, gr}$. Because $T_1$ and $T_3$ write to the same data item $e_{1,3}$, it follows that $\tau_{3, w} \prec \tau_{3, gr}$. Since no point is inserted between $\tau_{3, gr}$ and $\tau_{3, w}$, it follows that $\tau_{3, w} \prec_3 \tau_{3, gr}$. This contradicts the fact that $T_3$ reads 0 for $b_1$ in $\alpha_3$.**

Hence, $T_1 \notin com(\delta_5)$, and consequently $T_6$ reads 0 for $d_1$ in $\alpha_6$.

We next argue that $T_6$ reads 1 for $c_5$ in $\alpha_6$. Since $\delta_5 = \alpha_1 \cdot \alpha_2 \cdot \alpha_5 \cdot \alpha_6$ is legal and $I$ satisfies weak adaptive consistency, there exists a consistency partition $P(\delta_6)$ and a set $com(\delta_6)$ of committed and commit-pending transactions from $\delta_6$ which satisfy the conditions of Definition 5.3. We consider the following cases:

- **Assume first that $P(\delta_6)$ includes $G(T_6, T_6)$. Since the execution interval of any consistency group containing transaction $T_6$ must precede the execution interval of $G(T_6, T_6)$, it follows that $\tau_{6, w} < 6 \tau_{6, gr}$.**

- **Assume now that $P(\delta_6)$ includes $G(T_1, T_6)$ for some $i \in \{1, 2, 5\}$ and that $G(T_1, T_6) \in SI(P(\delta_6))$. Since $T_5 \prec_6 T_6$, it follows that $\tau_{5, w} < 6 \tau_{6, gr}$.**

- **Assume now that $P(\delta_6)$ includes $G(T_1, T_6)$ for some $i \in \{1, 2, 5\}$ and that $G(T_1, T_6) \in PC(P(\delta_6))$. Since $T_3$ reads 0 for $b_6$ in $\alpha_5$, it follows that $\tau_{6, gr} \prec_5 \tau_{6, w}$. Since no point is inserted between $\tau_{6, gr}$ and $\tau_{6, w}$, it follows that $\tau_{6, w} \prec_5 \tau_{6, gr}$. Because $T_3$ and $T_6$ write to the same data item $e_{5,6}$, it follows that $\tau_{5, w} < 6 \tau_{6, w}$. Since no point is inserted between $\tau_{6, gr}$ and $\tau_{6, w}$, it follows that $\tau_{6, w} \prec_6 \tau_{6, gr}$.**

We conclude that (in all cases) $\tau_{5, w} < 6 \tau_{6, gr}$. Thus, $T_6$ reads 1 for $c_5$ in $\alpha_6$.

Because $\alpha_6'$ and $\alpha_6''$ are indistinguishable to $pr_1$, it follows that $T_6$ reads the same values in $\alpha_6'$ and $\alpha_6''$. Since $T_6$ reads 2 for $b_2$ in $\beta_5'$, it follows that $\tau_{6, w} \prec \tau_{6, gr} \prec \tau_{6, gr}$. Because $T_2$ and $T_6$ write to the same data item $e_{2,5}$, it follows that $\tau_{6, w} < 6 \tau_{6, gr}$. Since $T_6$ reads 0 for $d_1$ and 1 for $c_6$, it follows that $\tau_{6, w} < 6 \tau_{6, gr} < 6 \tau_{6, gr}$. Thus, $\tau_{6, w} \prec \tau_{6, gr}$. Because $T_1$ and $T_6$ write to the same data item $a_1$, it follows that $\tau_{6, w} \prec \tau_{6, gr}$. Since $T_6$ reads 1 for $c_1$, it follows that $\tau_{3, w} \prec \tau_{6, gr}$. Thus, $\tau_{6, w} \prec \tau_{6, gr}$ in $\beta_5'$, and consequently, it follows that $T_7$ reads 1 for $a_6$ in $\beta_5'$ (see Figure 6) and in $\alpha_7$. **

Claim 4 states that $T_7$ reads 2 for data item $a_6$ in $\alpha_7$, and Claim 5 states that $T_7$ reads 1 for data item $a_6$ in $\alpha_7$. Since $\alpha_7$ is indistinguishable from $\alpha_7'$ to process $pr_7$ this is a contradiction. **

5. DISCUSSION

We proved the PCL theorem: in transactional systems it is impossible to ensure strict disjoint-access-parallelism (Parallelism), weak adaptive consistency (Consistency), and obstruction-freedom (Liveness). To circumvent the impossibility result it is sufficient to weaken just one of the three requirements. Weakening obstruction-freedom to a blocking liveness property makes it possible to ensure strict disjoint-access-parallelism and strong consistency (e.g. strict serializability) by using locks; these are the properties ensured by TL.[14] Likewise, weakening consistency makes it possible to ensure strict disjoint-access-parallelism and strong liveness. For example, allowing writes to the same data item to be viewed differently, as in PRAM consistency[28], makes it possible to trivially ensure strict disjoint-access-parallelism and wait-freedom, the strongest liveness property, without any synchronization between processes. In [11], we design a simple variant of DSTM[25], which satisfies snapshot isolation, obstruction-freedom, and the following weakening of strict disjoint-access-parallelism: two write operations on different data items contend on the same base object only if there is a chain of transactions starting with the transaction that performs one of these write operations and ending with the transaction that performs the other, such that every two consecutive transactions in the chain conflict. The PCL theorem shows that the distance between strict disjoint-access-parallelism and its non-strict forms draws a sharp line in the design of transactional systems.

Our theorem might at first glance look close to the CAP theorem[13] which states that it is impossible to ensure consistency, availability, and partition in a distributed system and weakening at least one of these requirements circumvents the impossibility result. In fact, they are different results. While consistency can be viewed as a safety property, and availability can be viewed as a liveness property, partition is not analogous to disjoint-access-parallelism. Specifically, partition tolerance ensures that the system tolerates arbitrary network partitions. Disjoint-access-parallelism on the other hand, does not ensure tolerance against failures but imposes that logical components of a system (transactions or operations) do not contend at low level (i.e. on base objects) if they do not conflict at high level (i.e. do not access the same data items).

Our definition of snapshot isolation is incomparable to strict serializability[9] and opacity[22]. This is because strict serializability and opacity are defined in terms of execution intervals whereas our definition of snapshot isolation is based on active execution intervals. The same holds for previous definitions of snapshot isolation, both in the database world[10], and in TM computing[4, 33]. We can easily remedy this problem by defining snapshot isolation in terms of execution intervals. Indeed, we did so in [11] and we were able to prove[11] that no TM implementation satisfies that version of snapshot isolation, obstruction-freedom, and strict disjoint-access-parallelism. This impossibility result[11] also holds if a primitive accesses up to $k$ base objects in one atomic step.
6. ACKNOWLEDGEMENTS

This work has been supported by the European Commission under the 7th Framework Program through the Transform (FP7-MC-ITN-238639) project and by the ARISTEIA Action of the Operational Programme Education and Lifelong Learning which is co-funded by the European Social Fund (ESF) and National Resources through the GreenVM project.

7. REFERENCES